Systematic Construction of Domains for Abstract Interpretation Frameworks

by

Ulf Nilsson
PhD theses:

*Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

( *Linköping Studies in Science and Technology. Dissertations.* )


No 97 Andrzej Lingas: Advances in Minimum Weight Triangulation, 1983.

*No 109* Peter Fritzson: Towards a Distributed Programming Environment based on Incremental Compilation, 1984.


Licentiate of engineering Theses:

( *Linköping Studies in Science and Technology. Theses.* )


No 73 Ola Strömfors: A Structure Editor for Documents and Programs. 1986.


No 108 Rober Bilos: Incremental Scanning and Token-based Editing. 1987


No 113 Ralph Rönnquist: Network and Lattice Based Approaches to the Representation of Knowledge. 1987.

No 118 Mariam Kamkar, Nahid Shahmehri: Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987

No 126 Dan Strömberg: Transfer and Distribution of Application Programs. 1987


Systematic Construction of Domains for Abstract Interpretation Frameworks

by

Ulf Nilsson
E-mail: urn@ida.liu.se

Abstract: The paper addresses two issues — initially sufficient conditions which ensure termination and correctness of abstract interpretations are formulated. Secondly we show how simple abstract domains which satisfy these conditions may be composed into more powerful and complex abstract domains which also satisfy the proposed requirements. We demonstrate the technique by constructing an abstract domain suitable for inferring (1) whether two variables are bound to terms with shared variables (2) whether a variable is bound to a ground term and (3) whether a variable is unbound or bound to another variable.

OTE

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden
Systematic Construction of Domains for Abstract Interpretation Frameworks

Ulf Nilsson
Department of Computer and Information Science
Linköping University
S–581 83 Linköping, Sweden

December 14, 1988

Abstract: The paper addresses two issues — initially sufficient conditions which ensure termination and correctness of abstract interpretations are formulated. Secondly we show how simple abstract domains which satisfy these conditions may be composed into more powerful and complex abstract domains which also satisfy the proposed requirements. We demonstrate the technique by constructing an abstract domain suitable for inferring (1) whether two variables are bound to terms with shared variables (2) whether a variable is bound to a ground term and (3) whether a variable is unbound or bound to another variable.

1 Introduction

Abstract interpretation (cf. [AH87,CC77,CC79]) is a general scheme which facilitates inference of e.g. run-time properties of programs. In the logic programming community this technique has gained substantial interest lately and several frameworks for abstract interpretation have been presented (cf. [Bru87,DW88,JS87,MU87,Mel87,Nil88]). Typical properties which can be inferred as a result of abstract interpretation include e.g. modes [Mel87,DW88,ST84], positions where occur-check may be safely omitted [Son86,Pla84] and types [HK88,BJ88]. Abstract interpretation also is of great importance as a preprocessor to partial evaluators [GCS88].

Abstract interpretation usually takes as a starting point some precise semantics of the program at hand. This semantics is often based on the standard operational semantics of logic programs, but it need not be — for instance,
[MS88a,MS88b] uses a bottom-up fixed point semantics. Unfortunately, this precise (or static) semantics cannot, in general, be computed in finite time. However, it may be safely approximated. This means that the static semantics is "contained" in an abstract semantics. This can be achieved by replacing the concrete domain by an abstract domain which contains less information than the concrete domain and by replacing operations on the concrete domain by abstract operations on the abstract domain.

In this paper two issues are discussed — on one hand we formulate some sufficient conditions under which the static semantics of a program may be effectively approximated. This results in some sufficient conditions on the abstract domain. Secondly, we show how simple abstract domains which satisfy these requirements may be composed into more complex and powerful abstract domains in such a way that the new domain also satisfies the requirements.

The rest of the paper is organized accordingly — Section 2 contains basic terminology and preliminaries, including lattice theory and a general background concerning abstract interpretation. In section 3 some sufficient conditions are formulated to guarantee termination of an abstract interpretation and that the static semantics is contained in the abstract semantics. Section 4 describes how abstract domains may be composed into new abstract domains and section 5 illustrates the approach through a small example. Section 6, finally, contains conclusions.

2 Preliminaries

2.1 The Kleene Sequence

We assume that basic notions from modern algebra and lattice theory are familiar. For introductory reading see e.g. [Bir67].

**Definition 2.1** A complete lattice is a poset \( \langle \mathcal{D}, \sqsubseteq \rangle \) where every \( X \subseteq \mathcal{D} \) has a least upper bound, \( \sqcup X \in \mathcal{D} \), and greatest lower bound, \( \sqcap X \in \mathcal{D} \). By \( \top \) and \( \bot \) we denote \( \sqcup \mathcal{D} \) and \( \sqcap \mathcal{D} \) respectively.

Denote by \( x \sqsubseteq y \) the fact that \( x \subseteq y \) and \( x \neq y \).

**Definition 2.2** Let \( \langle \mathcal{D}, \sqsubseteq \rangle \) be a complete lattice. The height of \( \langle \mathcal{D}, \sqsubseteq \rangle \) is the length of the longest sequence \( x_0 \sqsubseteq x_1 \sqsubseteq \ldots \) (The length of \( x_0 \) is 0, the length of \( x_0 \sqsubseteq x_1 \) is 1 etc).

**Definition 2.3** Let \( \langle \mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle \) be a complete lattice. The (ascending) ordinal powers of \( f : \mathcal{D} \rightarrow \mathcal{D} \) are defined thus

\[ f \uparrow 0 = \bot \]
\[ f \uparrow \alpha = f(f \uparrow (\alpha - 1)) \text{ for all successor ordinals } \alpha \]
\[ f \uparrow \alpha = \bigcup \{ f \uparrow \beta \mid \beta < \alpha \} \text{ for all infinite limit ordinals } \alpha \]

Specifically

\[ f \uparrow \omega \text{ designates } \bigcup_{i=0}^{\infty} f \uparrow i \]

The sequence \( f \uparrow 0, f \uparrow 1, \ldots \) is called the Kleene sequence of \( f \).

---

Tarski [Tar55] showed that if \( D \) is a complete lattice and \( f : D \to D \) is monotonic, then \( f \) has a unique least fixed point. Furthermore, due to Kleene [Kle52],

**Theorem 2.1** Let \( D \) be a complete lattice and \( f : D \to D \) continuous. Then \( f \uparrow \omega \) is the least fixed point of \( f \).

---

Finally we have the following lemmata.

**Lemma 2.1** Let \( \langle A, \leq, \lor, \land \rangle \) and \( \langle B, \subseteq, \cup, \cap \rangle \) be complete lattices and let \( f : B \to A \) be monotonic. Then for every \( X \subseteq B \) it holds that

\[ \lor \{ f(x) \mid x \in X \} \leq f(\cup X) \]

**Proof:** Clearly \( \forall x \in X (x \subseteq \cup X) \). Thus, due to the monotonicity of \( f \) we know that \( \forall x \in X (f(x) \leq f(\cup X)) \). Consequently

\[ \lor \{ f(x) \mid x \in X \} \leq f(\cup X) \]

---

**Lemma 2.2** Let \( \langle D, \subseteq, \cup, \cap \rangle \) be a complete lattice and \( f : D \to D \) a monotonic operation. Then \( f \uparrow \alpha \subseteq f \uparrow (\alpha + 1) \).

**Proof:** Use transfinite induction. For \( \alpha = 0 \) the proof is trivial since \( f \uparrow 0 \) is the bottom element.

Assume \( \alpha \) is a successor ordinal. By the induction hypothesis we have \( f \uparrow (\alpha - 1) \subseteq f \uparrow \alpha \). From the monotonicity of \( f \) we get that

\[ f \uparrow \alpha = f(f \uparrow (\alpha - 1)) \subseteq f(f \uparrow \alpha) = f \uparrow (\alpha + 1) \]
Finally, assume $\alpha$ is a limit ordinal. Then $f \uparrow \alpha = \cup \{ f \uparrow \beta | \beta < \alpha \}$. By the induction hypothesis and Lemma 2.1

\[
\bigcup \{ f \uparrow \beta | \beta < \alpha \} \subseteq \bigcup \{ f \uparrow (\beta + 1) | \beta < \alpha \}
\]

\[
\subseteq f \left( \bigcup \{ f \uparrow \beta | \beta < \alpha \} \right)
\]

\[
= f \uparrow (\alpha + 1)
\]

### 2.2 Abstract Interpretation

Roughly speaking, a program consists of a sequence $\ldots, S_i, S_{i+1}, \ldots$ of statements. Associated with each statement are two program points — one “to the left” and one “to the right” of each statement. In logic programming the body literals of clauses may be viewed as statements. This view is adopted in e.g. [Bru87] and [Nil88] whereas [JS87] have only one program point in each clause — at the “implication arrow”.

The idea of abstract interpretation is to associate with each program point, the set of possible execution states whenever control reaches the point. For now, we are not being precise as to what an execution state is — However, it is customary to define it to be a set of variable bindings — i.e. a (possibly partial) mapping from variables to data. Execution states will be called environments and sets of environments will be called contexts.

The context of each program point depends on contexts at other program points. Thus, for each program point we have an equation

$$\Theta_i = \Delta_i(\Theta)$$

where $\Theta$ designates a vector of contexts — each component of the vector consists of the context associated with one program point — and $\Theta_i$ denotes the context in $\Theta$ which is associated with program point $i^1$. Assume that a program with $n$ program points is given. Then finding a solution to the set

\[
\begin{cases}
\Theta_1 = \Delta_1(\Theta) \\
\vdots \\
\Theta_n = \Delta_n(\Theta)
\end{cases}
\]

of equations may alternatively be formulated as finding a fixed point of the mapping

$$cv(x) = (\Delta_1(x), \ldots, \Delta_n(x))$$

---

1. This is not entirely correct since some contexts may depend on input to the program. Such program points must be handled separately.
This fixed point (corresponding to a solution to the equations) need not be unique. Indeed it may not even exist. However, there are sufficient conditions that a least fixed point exists ([Tar55]) — namely if \( cv \) is monotonic.

**Definition 2.4** An abstract interpretation is a tuple \( \langle \mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top, cv \rangle \) where

- \( \langle \mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle \) is a complete lattice, and
- \( cv \) is a continuous mapping \( cv : \mathcal{D} \rightarrow \mathcal{D} \).

From Kleene's fixed point theorem we know that \( cv \uparrow \omega \) is the least fixed point of \( cv \). Given a program \( P \) we call \( cv \uparrow \omega \) the abstract semantics of \( P \). In the sequel we always assume that some \( P \) is given.

An abstract semantics of special interest is the so called static semantics (sometimes referred to as the collecting semantics) which provides exact information about the set of possible states (= environments) at each program point in the program.

Let \( \langle \mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top, cv \rangle \) be the static interpretation and, thus, \( cv \uparrow \omega \) the static semantics of some program \( P \). We say that the abstract interpretation \( \langle \mathcal{D}', \sqsubseteq, \sqcup, \sqcap, \bot', \top', cv' \rangle \) is a safe \( \gamma \)-approximation of the static semantics if \( \gamma : \mathcal{D}' \rightarrow \mathcal{D} \) is monotonic and

\[
cv \uparrow \omega \subseteq \gamma(cv' \uparrow \omega)
\]

In the following sections we shall see that the static semantics can be effectively approximated by some \( \gamma \) and some abstract interpretation.

## 3 Safeness and Termination

In this section we concern ourselves with the problem of finding some sufficient conditions to guarantee termination of abstract interpretations and safeness with respect to a static semantics.

The following theorem provides a sufficient condition to find the abstract semantics in finite time

**Theorem 3.1** Let \( \langle \mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top, cv' \rangle \) be an abstract interpretation. If \( \mathcal{D} \) is of finite height (wrt. \( \sqsubseteq \)), then there exists a finite ordinal \( n \) such that \( cv' \uparrow (n + 1) = cv' \uparrow n \).
Proof: Assume that the height is $n$. We know that

$$cv' \uparrow 0 \subseteq \cdots \subseteq cv' \uparrow (n + 1)$$

from Lemma 2.2. It should be clear that the sequence is not strictly increasing, i.e. $cv' \uparrow i \subseteq cv' \uparrow (i + 1)$ for $0 \leq i \leq n$, since the maximal increasing sequence only has length $n$.

To prove under what conditions the abstract semantics may be safely approximated we first define the concept of safe $\gamma$-approximation more precisely.

Definition 3.1 Let $\langle A, \leq, \lor, \land \rangle$ and $\langle B, \subseteq, \cup, \cap \rangle$ be complete lattices. Let $\gamma : B \rightarrow A$ be monotonic. We say that $f' : B^n \rightarrow B$ is a safe $\gamma$-approximation of $f : A^n \rightarrow A$ iff

$$\forall x_1, \ldots, x_n \in B^n \ ( f(\gamma(x_1), \ldots, \gamma(x_n)) \leq \gamma(f'(x_1, \ldots, x_n)))$$

We then have the following lemma.

Lemma 3.1 Let $f' : B \rightarrow B$ be a safe $\gamma$-approximation of a monotonic mapping $f : A \rightarrow A$. For any ordinal $\alpha$ it holds that $f \uparrow \alpha \leq \gamma(f' \uparrow \alpha)$.

Proof: Use transfinite induction. For $\alpha = 0$ the proof is trivial since $f \uparrow 0$ is the bottom element in $A$.

If $\alpha$ is a successor ordinal then assume that $f \uparrow (\alpha - 1) \leq \gamma(f' \uparrow (\alpha - 1))$. Clearly

$$f \uparrow \alpha = f(f \uparrow (\alpha - 1)) \leq f(\gamma(f' \uparrow (\alpha - 1))) \leq \gamma(f'(f' \uparrow (\alpha - 1))) = \gamma(f' \uparrow \alpha)$$

due to the monotonicity of $f$ and safeness of $f'$.

Finally if $\alpha$ is a limit ordinal

$$f \uparrow \alpha = \bigvee \{ f \uparrow \beta \mid \beta < \alpha \}$$

By the induction hypothesis we get

$$\bigvee \{ f \uparrow \beta \mid \beta < \alpha \} \leq \bigvee \{ \gamma(f' \uparrow \beta) \mid \beta < \alpha \}$$

and from Lemma 2.1

$$\bigvee \{ \gamma(f' \uparrow \beta) \mid \beta < \alpha \} \leq \gamma \left( \bigcup \{ f' \uparrow \beta \mid \beta < \alpha \} \right) = \gamma(f' \uparrow \alpha)$$

We are now in a position to establish the following sufficient condition.
Theorem 3.2 Let \( \langle D, \subseteq, \cup, \cap, \bot_D, T_D, \text{cv} \rangle \) be the static interpretation of a program and let \( \langle D', \leq, \lor, \land, \bot_{D'}, T_{D'}, \text{cv}' \rangle \) be an abstract interpretation. If \( \gamma : D' \rightarrow D \) is monotonic and if \( \text{cv}' \) is a safe \( \gamma \)-approximation of \( \text{cv} \) then \( \text{cv} \uparrow \omega \subseteq \gamma(\text{cv}' \uparrow \omega) \).

Proof: Follows directly from Lemma 3.1.

Corollary 3.1 If \( \langle D', \leq, \lor, \land, \bot_{D'}, T_{D'}, \text{cv}' \rangle \) is of finite height, then there exists a finite \( n \) such that \( \text{cv} \uparrow \omega \subseteq \gamma(\text{cv}' \uparrow n) \).

It is straightforward to find a safe approximation of a static semantics. Let \( \langle D, \subseteq, \cup, \cap, \bot_D, T_D, \text{cv} \rangle \) be the static interpretation of some program and let \( \langle D', \leq, \lor, \land, \bot_{D'}, T_{D'}, \text{cv}' \rangle \) be an abstract interpretation where

- \( D' = \{ T_{D'} \} \).
- \( \text{cv}'(x) = x \).
- \( \gamma = \{ T_{D'} \mapsto T_D \} \)

This proves the existence of an effective procedure which approximates the static semantics. However, this particular semantics is not very interesting. In practice we want to find an abstract interpretation where \( \text{cv}' \uparrow (n - 1) = \text{cv}' \uparrow n \) for relatively small \( n \) and where \( \gamma(\text{cv}' \uparrow n) \) contains as little overhead as possible wrt. the static semantics. These two goals are usually contradictory — in the abstract interpretation above \( n \) will be 1 but \( \gamma(\text{cv}' \uparrow 1) \) is the worst possible approximation of \( \text{cv} \uparrow \omega \). Thus, the main problem in abstract interpretation boils down to finding an abstract domain where granularity and height of the abstract domain are properly mixed.

4 Composing Domains

In this section we take the results from the previous section and develop a systematic method to construct abstract domains by composing smaller and less complex abstract domains into more powerful and expressive domains.

We know from the previous section that the domain should be a finite height complete lattice. Moreover there should be a monotonic mapping from the abstract domain to the concrete one.
Theorem 4.1 Let \((A, \subseteq, \cup, \cap)\) be a complete lattice. Furthermore, assume that \((A', \leq', \lor', \land')\) and \((A'', \leq'', \lor'', \land'')\) are complete lattices and that \(\gamma' : A' \rightarrow A\) and \(\gamma'' : A'' \rightarrow A\) are monotonic. That is
\[
x \leq' y \Rightarrow \gamma'(x) \subseteq \gamma'(y)
\]
\[
x \leq'' y \Rightarrow \gamma''(x) \subseteq \gamma''(y)
\]
Consider the product \((A' \times A'', \leq, \lor, \land)\) of the latter two where\(^2\)
\[
x \leq y \iff x_1 \leq' y_1 \land x_2 \leq'' y_2
\]
\[
x \lor y = (x_1 \lor' y_1, x_2 \lor'' y_2)
\]
\[
x \land y = (x_1 \land' y_1, x_2 \land'' y_2)
\]
\[
\gamma(x) = \gamma'(x_1) \land \gamma''(x_2)
\]
It then holds that \(\gamma\) is monotonic, i.e.
\[
x \leq y \Rightarrow \gamma(x) \subseteq \gamma(y)
\]

Proof:
\[
x \leq y \Rightarrow x_1 \leq' y_1 \land x_2 \leq'' y_2
\]
\[
\Rightarrow \gamma'(x_1) \subseteq \gamma'(y_1) \land \gamma''(x_2) \subseteq \gamma''(y_2)
\]
\[
\Rightarrow \gamma(x) \subseteq \gamma'(y_1) \land \gamma(x) \subseteq \gamma''(y_2)
\]
\[
\Rightarrow \gamma(x) \subseteq (\gamma'(y_1) \land \gamma''(y_2))
\]
\[
\Rightarrow \gamma(x) \subseteq \gamma(y)
\]

It is well known that the product of two complete lattices is itself a complete lattice. Moreover, if both lattices are finite height, then the product is finite height. Thus, the proposed strategy when constructing abstract domains is to start with a set \(D_1, \ldots, D_n\) of small trivial abstract domains with its own monotonic mapping \(\gamma_1, \ldots, \gamma_n\). Then a new domain \(D_1 \times \cdots \times D_n\) is constructed with a monotonic mapping \(\gamma(x) = \gamma_1(x_1) \land \cdots \land \gamma_n(x_n)\).

5 Extended Example

In this section we demonstrate how to compose two simple abstract domains into a more powerful one. To do this we first have to define what the concrete domain is. I.e. we have to define the concepts of environment, context and context-vector. Throughout this section we assume that a program \(P\) with \(n\) program points is given.

\(^2\)We denote the first and second component of \(x\) by \(x_1\) and \(x_2\) respectively.
5.1 A Concrete Domain

Let \( \mathcal{V} \) be the set of variables in \( P \) and let \( \mathcal{T} \) be the set of all terms. The set \( \mathcal{E} \) of environments is the set of all mappings
\[
\mathcal{E} = (\mathcal{V} \rightarrow \mathcal{T})
\]

The sets \( \mathcal{C} \) and \( \mathcal{CV} \) of contexts and context vectors are
\[
\mathcal{C} = \varphi(\mathcal{E})
\]
\[
\mathcal{CV} = \underbrace{\mathcal{C} \times \cdots \times \mathcal{C}}_n
\]

Clearly, \( (\mathcal{C}, \subseteq, \cup, \cap) \) is a complete lattice. Also \( (\mathcal{CV}, \subseteq, \cup, \cap) \) is a complete lattice where the operations apply component-wise.

5.2 Abstract Domain I

Next we construct a set of abstract contexts. First we introduce the set \( \{v, t\} \) with partial order \( v \leq t \) and the set \( \mathcal{C}' \) of abstract contexts
\[
\mathcal{C}' = (\mathcal{V} \rightarrow \{v, t\})
\]

ordered accordingly
\[
\phi_1 \leq \phi_2 \iff \forall x \in \mathcal{V} \ (\phi_1(x) \leq \phi_2(x))
\]

The mapping \( \gamma' : \mathcal{C}' \rightarrow \mathcal{C} \) is defined thus
\[
\gamma'(\phi) = \{ \theta \in \mathcal{E} \mid (\phi(x) = v) \Rightarrow \text{variable}(\theta(x)) \}
\]

where \( \text{variable}(t) \) holds iff \( t \) is a variable. Hence, intuitively \( \{\ldots, x \mapsto v, \ldots\} \) means that \( x \) is always bound to a variable or is unbound.

It is easy to show that \( \gamma' \) is monotonic and that \( (\mathcal{C}', \subseteq, \lor, \land) \) is a complete lattice where \( \lor \) and \( \land \) are defined in the obvious way.

**Example 5.1** Consider a program containing only the variables \( x \) and \( y \). The lattice of abstract contexts is depicted in Figure 1.
5.3 Abstract Domain II

Now, let us construct a second abstract domain. Let $C''$ be the set of all symmetric subsets of $(\mathcal{V} \times \mathcal{V})$. It is easy to verify that $(C'', \subseteq, \cup, \cap)$ is a complete lattice under set inclusion and that

$$\gamma''(R) = \{ \theta \in \mathcal{E} \mid ((x, y) \not\in R) \Rightarrow \text{indep}(\theta(x), \theta(y)) \}$$

is monotonic (indep($t_1, t_2$) holds iff. the sets of variables in $t_1$ and $t_2$ are disjoint). Intuitively, if $(x, y)$ is contained in an abstract context then $x$ and $y$ may be bound to terms which contain a common variable. If $(x, y)$ is not contained in the context, then $x$ and $y$ are always independent (i.e. contain no common variables). Notice that $x$ must be ground if $(x, x)$ is not contained in a context.

Example 5.2 The domain of abstract contexts for a program with two variables, $x$ and $y$, is depicted in Figure 2. 

\[\text{Example 5.2}\]
5.4 Abstract Domain III

We now take the previous two abstract domains and compose them into a new domain of abstract contexts.

Given that

\[ x \sqsubseteq y \iff x_1 \leq y_1 \land x_2 \subseteq y_2 \]
\[ x \sqcup y = (x_1 \lor y_1, x_2 \cup y_2) \]
\[ x \sqcap y = (x_1 \land y_1, x_2 \cap y_2) \]

Then \( (C' \times C'', \sqsubseteq, \sqcup, \sqcap) \) is a complete lattice. And the mapping

\[ \gamma(x) = \gamma'(x_1) \cap \gamma''(x_2) \]

is monotonic by Theorem 4.1.

Example 5.3 Below follow some examples of abstract contexts and their meaning in terms of concrete contexts wrt. \( \gamma \).

\[ \{x \mapsto v, y \mapsto t\}, \{(x, x)\} \]
Here \( y \) must be bound to a ground term (since \( (y, y) \) is not contained in the second component). \( x \) is unbound or bound to another variable.

\[ \{x \mapsto v, y \mapsto t\}, \{(y, y)\} \]
According to the first component \( x \) is always bound to a variable but according to the second it is always bound to a ground term. Thus, the expression denotes the empty set of environments. Practically this means that this context is associated with a dead piece of code.

\[ \{x \mapsto t, y \mapsto t\}, \{(x, y), (y, x)\} \]
From the first component we see that \( x \) and \( y \) may be bound to any term. From the second we conclude that they are both ground.

6 Conclusions

Some sufficient conditions were formulated to guarantee (1) termination of abstract interpretation and (2) safeness with respect to a static semantics.

We also proposed a systematic approach to construction of domains for abstract interpretation frameworks. To our knowledge no such method has been formulated but domains have been constructed in a rather ad hoc manner relying more or less on the ingenuity of the authors. In [JS87] the approach is used in one of the examples but without any discussion as to whether the result satisfies
the required properties of the domain. The approach suggested above has two
definite advantages over ad hoc construction of domains

- Construction of relatively complex domains becomes easier by a stepwise
  composition of smaller domains, and

- verification that an abstract domain is a complete lattice with a monotonic
  mapping from the abstract domain to the concrete one is not necessary
  since it follows from the fact that composed domains automatically satisfy
  the requirements if the smaller domains do.

Development of an abstract interpretation should start with the construction
of a static interpretation and static semantics (call it $cv \uparrow \omega$) for the desired
programming language. Then for each application of abstract interpretation, an
abstract domain is built — a domain which satisfies the following conditions

- it is a finite height, complete lattice.

- there is a monotonic mapping $\gamma$ from the abstract domain to the concrete
  one.

Finally this abstract domain is augmented with a continuous operation which is
a safe $\gamma$-approximation of $cv$.

In general this process requires a great deal of work. However, the first
step (construction of a static semantics) need only be performed once for every
programming language. The main contribution of this paper is the reduce the
work needed in the second step. However, the most laborious step is the last
one. At present we do not foresee any simple method to automate this process
but still have to rely on the skill and ingenuity of the designer.

Acknowledgements

Thanks are due to Jan Maluszyński for his continuous support and to Staffan
Bonnier for invaluable comments. Fundings were provided by the Swedish Na-
tional Board for Technical Development (STU and STUF).

References

[AH87] S. Abramsky and C. Hankin, editors. Abstract Interpretation of Declar-


Systematic Construction of Domains for Abstract Interpretation Frameworks

Author: Ulf Nilsson

Abstract: The paper addresses two issues — initially sufficient conditions which ensure termination and correctness of abstract interpretations are formulated. Secondly we show how simple abstract domains which satisfy these conditions may be composed into more powerful and complex abstract domains which also satisfy the proposed requirements. We demonstrate the technique by constructing an abstract domain suitable for inferring (1) whether two variables are bound to terms with shared variables (2) whether a variable is bound to a ground term and (3) whether a variable is unbound or bound to another variable.
A Selection of Previous Research Reports.


LiTH-IDN-R-88-34 Arne Jönsson, Nils Dahlbäck: Talking to a computer is not like talking to your best friend. Also in Proc. of the First Scandinavian Conference on Artificial Intelligence, March 9-10, 1988, Tromsø, Norway.


LiTH-IDN-R-88-30 Erik Sandewall: Future Developments in Artificial Intelligence. Also in Proc. of European Conference on Artificial Intelligence (ECAI), Munich, August, 1988.


LiTH-IDN-R-88-18 Nils Dahlbäck: Mental Models and Text Understanding - a Commented Review.
The Department of Computer and Information Science
Linköping University

organizes undergraduate and graduate studies in Computer Science, Telecommunication and Computer Systems, and Administrative Data Processing. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- **ACTLAB - Laboratory for Complexity of Algorithms**, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.

- **ASLAB - Application Systems Laboratory**, which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.

- **CADLAB - Laboratory for Computer-Aided Design of Electronics**, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.

- **LIBLAB - Laboratory for Library and Information Science**, which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.

- **LOGPRO - Laboratory for Logic Programming**, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.

- **NLPLAB - Natural Language Processing Laboratory**, which conducts research related to the development and use of natural language interfaces to computer software.

- **PELAB - Programming Environments Laboratory**, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.

- **RKLLAB - Laboratory for Representation of Knowledge in Logic**, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

**Research Reports 1988**


LiTH-IDA-R-88-44 Ulf Nilsson: Towards a Framework for the Abstract Interpretation of Logic Programs.

